SOFTWARE MAINTENANCE: RESEARCH AND PRACTICE, VOL. 8, 345–356 (1996)

Book Review

Improving the Maintainability of Software

RICHARD WEST

CCTA, London: HMSO, 1993, 104 pages

SUMMARY

The monograph, *Improving the Maintainability of Software*, has the objective of providing practical guidance on the maintainability of software: maintainability definitions, factors, building and improvement suggestions, and an assessment method.

The first part of this review provides an introduction to the monograph and discusses the general software maintainability guidance it provides. The second part provides a more in-depth review of the maintainability assessment method.

This monograph is easy to read and understand. The guidance provided appeals to common sense and reflects an understanding of many software maintenance concerns. Although the scope of the monograph is primarily business information systems, the guidance is basically sound for many other types of applications in business, industry, government and military organizations. The defined framework of maintainability factors, features, and measures provides interesting guidance for discovering potential areas of concern. Identifying these areas of concern can be a basis for directing action and resources to the more critical maintenance activities. The maintainability assessment method provided by this monograph is not adequately supported by references or detailed information.

Definitely read this monograph for its practical guidance. But be careful when applying the maintainability assessment method and analysing the results.

KEY WORDS: maintainability; risk assessment; complexity; software metrics; software assessment; software features; software factors

1. OVERVIEW OF MONOGRAPH

Background

The stated purpose of this monograph is to provide practical guidance on the maintainability of software. In particular, this monograph:

- defines maintainability and explains why it is important;
- describes the factors which contribute to the maintainability of software under development, and of existing software systems;
- gives guidance on building for and improving maintainability; and
- provides a simple approach for assessing maintainability.

CCC 1040–550X/96/050345–12

Received 12 March 1996
Revised 29 May 1996

The basis for the results is a study carried out on behalf of CCTA, the British Government Centre for Information Systems. The study included primarily business information system applications. The study included mainframe, personal computers, and distributed systems with batch or on-line processing. The monograph audience includes project managers, software developers and maintainers, and persons who license or contract software from a third party.

This monograph is one volume in an Information Systems Engineering Library developed by CCTA. The Information Systems Engineering (ISE) library is part of a complete Information Systems approach. Information Systems Guides and Information Technology Infrastructure Library volumes document this approach. Other publications specifically supplement this monograph. Publication topics include: management of software maintenance; change management; configuration management; help desk; problem management; service level management; software control and distribution; software lifecycle support; testing for operational use; and third party and single source maintenance.

This monograph has a main body (52 pages) and several annexes (52 pages). The main body provides the background, definitions, descriptions and qualitative guidance on maintainability factors, and an introduction to developing, preserving, improving, assessing, and monitoring the maintainability of software systems. Two annexes provide details necessary to complete software maintainability assessment forms for a Maintainability Measure (MM) and a Maintainability Index (MI). Example Maintainability Plans, actually more like maintainability improvement descriptions, are in a third annex. A brief description of the process and activities of software maintenance is in a fourth annex. A Bibliography of CCTA publications, a short list of other pertinent publications, and a concise Glossary of terms conclude the monograph.

What is maintainability?

There are many interesting tidbits of guidance provided in this monograph. The following statement was particularly encouraging.

'Software maintenance is an activity in its own right. . . . and maintenance work should not be regarded as merely an offshoot of development.'

This recognition is refreshing since too often other literature hides the criticality and uniqueness of software maintenance within statements like 'software maintenance is just a continuation of software development'.

Software maintainability attributes and software maintenance activities are clearly differentiated. Maintainability attributes affect the ease of conducting the maintenance activities. Maintenance activities follow a unique maintenance concept. Software maintainability attributes facilitate efficient balancing of software maintenance cost, schedule and quality.

The five steps of software maintenance are change analysis, identification, implementation, test and distribution. The monograph could have provided more information on how software maintenance tasks differ from software development tasks. The specific task activities conducted during the five steps have distinct maintenance objectives. The maintenance focus is on specific problem solving—from the 'small to the large'. Problems identify small areas in the software that must be understood and changed within the context of the whole system. With adequate product maintainability attributes, the scope of

changes required to solve the problem is minimized. With adequate process maintainability attributes, the changes can be cost-effectively implemented and product quality improved.

The software maintainability factors derived from the CCTA survey study can be grouped under product, process, and resource categories. A somewhat paraphrased factor list is illustrated in Figure 1.

Unfortunately, little information is presented and no references included, on the survey data that supports how these factors were derived. It is important to understand the quantitative accuracy of the data upon which the factor identification is made. Later this deficiency is discussed relative to application of the maintainability assessment methods.

Developing maintainable software systems

The monograph emphasizes planning for software maintenance during both development and support phases. Desirable maintainability attributes must be a part of the development process and resulting products. If not present, such attributes must be incorporated into the maintenance process and new software releases through maintainability improvements. Sample maintainability plans are provided in an annex. The two example plans illustrated there are actually more like maintainability process improvement plans and should not be confused with maintenance activity plans. One example maintainability plan objective, 'the target new system is to reduce maintenance costs by 40 per cent', actually addresses how the development processes are going to be adapted to ensure the objective is met. A maintenance activity plan would more typically specify the tasks to:

- identify maintenance activity profiles;
- determine needed resources to accomplish those activities;
- acquire the resources prior to the maintenance phase;
- transition from the product development phase to the product maintenance phase; and
- conduct the maintenance activities during the support phase.

Typical software development activities such as requirements, design, implementation,

Major Software Maintainability Factors

Product (Software System)
— size; complexity; structuredness; age; documentation

Development Processes and Resources
— maintainability as an objective; standards and methods; schedule constraints;
— development team skills, attitude, experience, motivation
— relationships with customers, users, operations, supporters, third party suppliers
— environment for development and test

Maintenance Processes and Resources
— change process for problem analysis, identification, implementation, release
— maintenance team skills, attitude, experience, motivation
— relationships with customers, users, operations, supporters, third party suppliers
— environment facilities for problem replication, live data access, rapid response, change

Figure 1. Major software maintainability factors

and test are discussed from the perspective of achieving maintainable products. Adherence to standards with maintainability objectives and use of maintainability attributes as a measure of delivered quality get attention. Most of this information has appeared in the literature, but is nicely summarized in the monograph.

One valuable comment on packaged software (i.e., commercial-off-the-shelf—COTS) is worth illustrating, since its maintenance impact is not always well understood by management or software personnel.

'Although packaged software systems may be simple and inexpensive to implement, they can present problems during the maintenance phase if changes necessitated by a business change cannot be fully implemented. There are several factors which affect this, including a lack of flexibility in the package, errors contained in the package, non-availability of source code, limited documentation and the requirement to keep up with new releases.'

The direction today is towards using commercial packages to accomplish as much operational functionality as possible in order to reduce the development cost. The warning is that this could create cost problems during maintenance—and even jeopardize the complete business enterprise. The requirement to keep up with new releases affects not only operational software, but the support systems as well. As an example, for each COTS software package that is part of the maintenance environment or the operational system software, there will be maintenance tasks required with each new software package release or upgrade. The choice could be to not integrate the updated packages, but vendors frequently refuse to support old versions of their software—or at least lose the capability to do so. Suppose as a minimum each new package release requires one person day for each of these activities:

- analyse the release impact;
- install the release on a test system;
- run regression tests to ensure there are no 'show stoppers';
- update the documentation on the system configuration for customer use; and
- release the updated product to customers.

Typically, a new software version release includes several changes in order to reduce the frequency of updates and the impact on the customers. Thus, there is a multiplicative effect on the complexity of the releases and the time to accomplish each of the above activities. One system analysed by this reviewer was to include over 200 commercial packages for use in the operational system and maintenance environment. These packages included a secure database system, a secure operating system, operational user interface generators, analysis tools, translators, and desk top publishing systems. Several full-time equivalent personnel, as well as significant facility and support system resources, could be consumed each year just to support the implementation of commercial software upgrades. Be aware, there are software maintenance costs besides the maintenance support agreement with the vendor. If the packaged software is a critical part (e.g., the secure database management system) of the operational software and the vendor goes out of business, there can be critical repercussions on the user's whole business enterprise.

Preserving and improving the maintainability of software

This monograph describes the usual corrective, perfective, and adaptive maintenance tasks. In addition, preventive maintenance is singled out as an important part of improving the maintainability of software. The other three types of maintenance normally include such preventive tasks, but it can be important to isolate the cost and benefits of improving the capability of the software to be maintained. Maintainability improvement projects can be an effective use of limited maintenance resources to reduce future maintenance problems.

The general approach promoted by the monograph for monitoring software maintainability is very similar to most 'plan, do, check, act' methods. Maintainability concerns are identified through an assessment that determines a Maintainability Measure and a Maintainability Index. An analysis of the assessment results identifies important concerns and prioritized actions. The maintenance organization develops a Maintainability Improvement Plan to address the highest priority actions and conducts preventive maintenance actions in accordance with the plan. A cost/benefit analysis of the improved maintainability provides feedback. This approach supports the current quality emphasis on software process improvement, but with a specific focus on maintenance. Most current process improvement methods focus on development processes and largely ignore anything to do with software maintenance.

The primary contribution of this monograph to maintainability process improvement is the maintainability assessment method. This method establishes a baseline of maintainability attribute measures and maintenance task capabilities of the organization. The Maintainability Measure is computed by summing weighted evaluation scores on software maintainability attributes for the main factors identified by the CCTA research study. A score sheet is provided along with a short discussion of each factor. The Maintainability Index measures error resolution time, resolution result, change time and change result—in essence, the productivity and quality of the actual maintenance activity. These two measures can be used to compare with other evaluated software scores. Specific areas of concern can be highlighted relatively easily.

A small paragraph is included on 'reaping the benefits'. It is essential that software maintainability improvement activities document the benefits and provide a forum for ensuring the benefits and cost are reasonable and recognized. This monograph discusses a few of the benefits. More supporting information on these topics would have been useful.

2. TECHNICAL BASIS FOR ASSESSMENT METHOD

Two metrics

Annexes A and B contain details of the software maintainability assessment method that is the technical basis for the monograph's guidance. These annexes explain the Maintainability Measure and Maintainability Index computation and use. This monograph states:

'The guidance given in this volume draws on the results of a study carried out on behalf of CCTA, the aim of which was to identify factors which affect maintainability and to assess

their relative importance. A strongly practical approach was adopted, and information was gathered on 35 current application software systems from eight organizations in the following business sectors: central government; life insurance; credit card processing; retail accounting; and research and development.

In six out of the eight organisations, application maintenance support was provided by a third party supplier. Mainframe, PC and distributed systems were included in the study as well as on-line and batch processing systems. No defence, command and control or safety critical systems were examined. Systems maintenance staff provided information on the maintainability factors which they considered to be important, and detailed questionnaires were then completed for the chosen systems providing information on maintainability factors and current activity. The relative maintainability of these systems was assessed by reference to data on the extent and success of recent maintenance activity. Statistical analyses of these data highlighted the factors which had the greatest impact on the maintainability of the systems.'

This information is very important because it provides the application scope and essentially the only description or reference of the study effort upon which the maintainability assessment is based. It would be very interesting to read about the study survey, organization characterizations and statistical analysis methods. A detailed technical report on this study was not referenced by the monograph.

The monograph specifically indicates that no defence (military) systems were examined. The references (Peercy, 1981; Peercy *et al.* 1985, 1986a, 1986b) provide defence systems research that parallels results in this monograph. In the latter research, a Risk Assessment Methodology for Software Supportability (RAMSS) was derived over a period of almost 12 years. RAMSS is based on extensive maintenance data and evaluations from United States Air Force software maintenance support centres. The RAMSS defines two main risk values: a Maintainability Evaluated Risk and Maintenance Estimated Risk. The RAMSS Maintainability Evaluated Risk is a measure of a software's maintainability characteristics for products, processes and resources. The Evaluated Risk is compared with a database of over 80 software system evaluations. The Evaluated Risk is very similar to the monograph's Maintainability Measure. The RAMSS Maintenance Estimated Risk is a measure of a software maintenance organization's capability to accomplish successfully a profile of maintenance tasks (corrective, perfective and adaptive) with specified support personnel and support systems. The Estimated Risk is compared with a database of over 300 maintenance block releases. The Estimated Risk is very similar to the mongraph's Maintainability Index.

Maintainability measure (MM)

Worksheet

The Maintainability Measure identifies whether software processes, products and resources have the attributes that make the software maintainable. A lower score indicates better maintainability. The Maintainability Measure is a composite score in the range one to 821·5. (The lower range actually appears to be 10, not one as the monograph states, since the complexity factor is at least nine and the age factor is at least one.) The Maintainability Measure has 20 major factors. Each factor is defined by a set of feature

attributes. Each feature is scored in accordance with a worksheet matrix. Each feature has a raw score and each factor has a weight. The product of the feature raw score and factor weight is the feature weighted score. The sum of the feature scores is the factor score. The sum of the factor scores is the Maintainability Measure.

Table 1 gives a partial example of the Maintainability Measure for the complexity factor. There are several basic problems with this approach that have not been adequately explained in the monograph or references to the research study. Some of these problems are discussed below.

Factor/feature/measure

The factors and associated features are described in some detail, but there is little information explaining how these factors and associated features were derived. Earlier in the monograph body (page 15) there is a list of the factors 'found in the CCTA study to be the most important.' Although the 'factors' listed are integrated into Annex A, the integration is sometimes at the factor and sometimes at the feature level. In addition, there is no understanding of why the features are given the assigned values. Presumably, the CCTA study results, or perhaps subsequent analysis, derived the values. For example, in Table 1 for the factor 'Complexity', the code complexity feature is given a maximum value of 2·0 and data structures feature has a maximum value of 1·5. This implies that code complexity may be more influential than data structuring for software complexity. A feature's range of values is sometimes continuous, sometimes based on a yes/no, and sometimes is more of a discrete set of values. In the case of code complexity is it suggested that either McCabe's cyclomatic complexity or Halstead's difficulty metric be used as a base and then converted by proportion to the range 0 to 2. Since neither the McCabe nor Halstead metrics have a maximum value, the monograph's suggestion is technically not possible. However, an organization could certainly define a conversion mapping using some reasonable grouping of values (e.g., for McCabe's cyclomatic complexity, 0–100 maps to 0–2, with any values over 100 mapping to 2). The validity of such a mapping is hard to determine (Zuse, 1991). If Halstead's difficulty metric were chosen, then there are additional complications. The code complexity overlaps with the data structure complexity since Halstead's measures are based on operands (data) as

Table 1. Example MM: complexity factor score

Factors	Features	Maximum score	Feature score	Weighted score
Complexity		Weight = 9		
	Business requirement	2·5	0	0
	Application	0·5	0	0
	Data structures	1·5	1·5	13·5
	Several languages used	1·5	1·5	13·5
	Code complexity	2·0	0·5	4·5
	Distributed site	0·5	0·5	4·5
	Customized software	1·5	0	0
	Factor total		4	36

well as operations (Halstead, 1977). This overlap makes it more difficult to isolate feature meaning.

Measurement scale and weights

It is very difficult to define measurement scales precisely that retain the desirable relations and operations of the factors and features (Fenton, 1991; Zuse, 1991). When deriving a factor/feature/measure hierarchy, it is important to ensure that the measurement scale is the same. It is important to ensure that the operations and relations, such as arithmetic and ordering, are valid if used. In the example above for the complexity factor, it can be seen that there are several different scales of measure used for the features. The higher score for a feature is supposed to mean that the software complexity is worse, and that the software's maintainability is also worse. The assumption is that feature scores can be summed to arrive at a complexity score. In the analysis section on assessment results (Section 6.3.2) there is the additional assumption that once the Maintainability Measure scores are computed for all systems that the mean of the scores can be computed. These assumptions imply (Fenton, 1991), that the measurement scale must be a ratio scale so ratios make sense and averages can be computed. All measurement scales must be the same (or capable of being mapped to a common scale), the intervals on the measurement scale must represent the same range of attribute behaviour, and there must be a 'zero'. For example, an increase from 0 to 1 in code complexity must be the same as an increase from 1 to 2. The decrease in software maintainability as the Maintainability Measure increases from 100 to 200 must be the same as the decrease in software maintainability as the Maintainability Measure increases from 700 to 800.

The measurement scales do not appear to be ratio or interval. A decrease in software maintainability for scores from 100 to 150 is not at all likely to be the same as from 200 to 250 or from 750 to 800. Furthermore, the measurement scale may not be ordinal, i.e., preserves order in the normal interpretation. As an example, Table 2 indicates the complexity feature values for two different systems. Is the complexity of System 2 really worse than the complexity of System 1?

This argument can easily be extended to all the factors and the overall Maintainability Measure. The question is: given two software systems S_1 and S_2 with respective Main-

Table 2. Example MM: complexity factor scores for two systems

Factors	Features	Maximum score	Feature score System1	Weighted score System 1	Feature score System 2	Weighted Score System 2
Complexity		Weight = 9				
	Business requirement	2·5	0	0	1·0	9·0
	Application	0·5	0	0	0·5	4·5
	Data structures	1·5	1·5	13·5	1·0	9·0
	Several languages used	1·5	1·5	13·5	1·0	9·0
	Code complexity	2·0	0·5	4·5	0·5	4·5
	Distributed site	0·5	0·5	4·5	0·5	4·5
	Customized software	1·5	0	0	0	0
	Factor total		4	36	4·5	40·5

tainability Measure scores of 400 and 450, is it really true that system S_2 is really more difficult to maintain than S_1? The feature values, factor weights and measurement scales must have a strong basis derived from the research study results. This monograph does not provide such evidence.

Maintainability index (MI)

The Maintainability Index represents the capability of the software maintenance organization to implement changes to the existing software products using existing facilities and support systems. The Maintainability Index is a composite score in the range of 400 to 1600 (again, the lower score is better). The Maintainability Index has four major factors: error resolution time; resolution result; change time; and change result. Each factor has a range of possible response times or impact results. Each response time or impact result has its own unique weight. A percentage of the changes applies to each response time or impact result. The response time or impact result score is the product of the percentage times the weight. The factor score is the sum of the weighted scores. The sum of the factor scores is the Maintainability Index.

Table 3 gives a partial example of the Maintainability Index factor and response/impact answer score. There are several basic problems with this approach that have not been adequately explained in the monograph or by references to the research study. These problems are very similar to those briefly covered in the discussion about the Maintainability Measure. In addition, it is clear that the time scale for correcting errors is not orientated to a 'block-release' concept. In a 'block-release' concept, an updated version of software is released on a somewhat regular basis (usually not every day or week). This updated version of software contains changes to correct errors, improve capability and adapt to possible changes in the environment. It is not clear that continuous releases of the software is a good maintenance concept. Users do not want to be interrupted daily with changes that may or may not affect them. On the other hand, if a critical change needs to be implemented, then it should be possible to respond within a short time period. The maintenance concept for user support is very dependent on the software application, the user community, and the software system. Clearly, Table 3 encourages a crisis-mode, emergency response concept which may not be the best way to promote software maintenance improvements. Or, it may reflect a sequential handling of maintenance tasks to completion, but with a collection of them into 'bundles' or 'packages' to be held for later block release. During early software system implementation, the continuous release

Table 3. Example MI: error resolution time score

Error resolution time*	Observed %	Weight	Score
Errors corrected in:			
Less than 2 hours	10	1	10
2 hours – 1 day	30	2	60
1 day – 2 days	50	3	150
2 days or more	10	4	40
	Sub-total A:		260

*Includes effort to diagnose, test, and implement (including management).

scheme may be useful for assessing emergency response capability or help desk response. The more normal block release implementation, including full implementation of pertinent emergency changes, could be assessed using another set of response times.

Assessment analysis using MM and MI

Once the MM and MI have been computed for the software systems of an organization, the monograph suggested that these two numbers be plotted on a diagram that resembles a four-quadrant scatter plot with the X–Y axes being the MM and MI scales, respectively. It is suggested that the intersection of the axes is organization-specific. That is, the intersection is determined by using the mean values of MM and MI across all of the organization's software systems. For example, suppose (average MM, average MI) = (400 800) is the intersection as shown in Figure 2.

Besides the various measurement difficulties described in the MM and MI paragraphs above, there are some other interesting analysis problems. It is not clear that being in any one of the four suggested quadrants might provide much information as to prioritization of which systems to improve. Customer needs are probably going to dictate which systems receive prioritized attention—but, the improvements still should be reflected through the assessment measures. The points P1 to P4 in the legend illustrate the problem. Although P1 (for system 1) is in the worst quadrant, it does not seem to be all that much worse than P3 (for system 3) that is in the best quadrant. The importance of assessing the factors and features is that the assessment can help identify areas of maintenance, such as data structure usage or time to correct errors, that can be directly targeted for improvement.

It might be better to set the intersection of both axes at the minimum points of the MM and MI scales and construct a simple scatter plot (with one quadrant). Central tendencies of the scatter plot would not need the measurement scales to be interval, only ordinal. Outliers can then be more easily identified.

The monograph does discuss some technical difficulties and provides guidance in the assessment results analysis section, Section 6.3. Consistently applying the factors, features, measure values and weight values within an organization is mentioned. Comparing

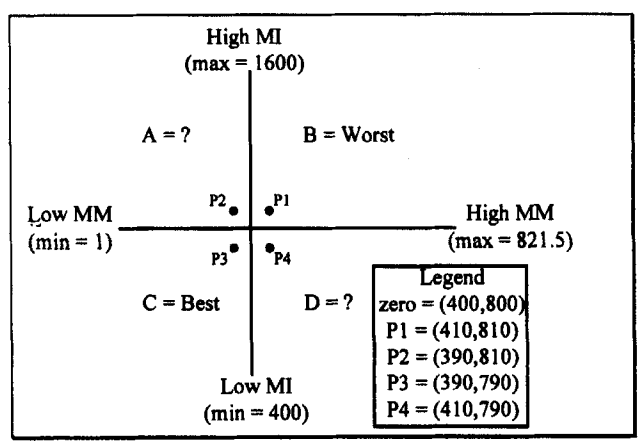

Figure 2. Example maintainability plot

assessments across different organizations is discouraged. The numbers for MM and MI are not the most helpful aspect—the individual elements which contributed to those numbers are where remedial action might be focused. It is suggested that the results are somewhat subjective and must be interpreted intelligently if true benefit is to be derived.

A few minor suggestions relate to the scales and naming conventions. Recall the point noted earlier: that the monograph adopts the convention that a lower value indicates better maintainability. It is usually desirable that the name selected for the top level, factor or feature implies the increasing or decreasing aspect of the measurement scale. For example, a higher Maintainability Measure should imply better, not worse maintainability. A similar statement holds for the Maintainability Index. Use of the term complexity as a factor implies that the higher scores should mean higher complexity and worse maintainability—which is true. However, higher scores for structuredness should imply better structure and better maintainability—which is not true. As another naming convention suggestion, since the Maintainability Index indicates the maintenance activity time and effort of the organization, it might have been better to use the name Maintenance Index. This name would have provided a little more distinction from the Maintainability Measure.

3. SUMMARY AND CONCLUSIONS

The purpose of this review has been to assist the reader in determining whether this monograph is worth reading and why. This monograph is definitely recommended reading for all audiences with an interest in software maintainability. The software maintainability and maintenance activity guidance is practical and easy to use. Information systems are the primary focus, but most of the general guidance applies to scientific systems as well.

However, caution is recommended in the interpretation of the results of the monograph's assessment method. The technical basis of the maintainability assessment method, including the two primary indicators—the Maintainability Measure (MM) and the Maintainability Index (MI)—needs to be more thoroughly discussed or at least supported by references. A detailed technical report containing the sanitized data from the CCTA study should be referenced and available. The report should describe the analysis methods and derivation of the software maintenance factors, features, and weights. In the absence of such supporting data, the user of the monograph's method should gather and analyse data from trying the monograph's assessment method in the user's own environment.

The Maintainability Measure, the Maintainability Index, and the dependencies between these two entities are important and should be further studied and documented.

DAVID E. PEERCY
Sandia National Laboratories
Albuquerque, NM, U.S.A.

References

Fenton, N. E. (1991) *Software Metrics: A Rigorous Approach*, Chapman & Hall, London.

Halstead, M. H. (1977) *Elements of Software Science*, Elsevier North-Hollard, Inc., New York, NY.

Peercy, D. E. (1981) 'A software maintainability evaluation methodology', *Transactions on Software Engineering*, **SE-7**(4), 343–351.

Peercy, D. E. and Tomlin, E. (1987) 'Assessing software supportability risk: a minitutorial', *Proceedings of the Conference on Software Maintenance—1987*, IEEE Computer Society Press, Austin, TX, pp. 72–80.

Peercy, D., Huebner, W., Estill, M. and Wu, J. (1985) *Software Supportability Risk Assessment in OT&E: Historical Baselines for Risk Profiles*, Volumes I and II, BDM/A-85–0510-TR, BDM, Inc., Albuquerque, NM.[1]

Peercy, D. and Huebner, W. (1986a) *Risk Assessment Methodology for Software Supportability (RAMSS): Guidelines for Adapting Software Supportability Evaluations*, BDM/ABQ-86–0090-TR, BDM, Inc., Albuquerque, NM.[1]

Peercy, D., Estill, M. and Shaw, K. (1986b) *Risk Assessment Methodology for Software Supportability (RAMSS): User's Handbook*, BDM/ABQ-85–1270-TR, BDM, Inc., Albuquerque, NM.[1]

Zuse, H. (1991) *Software Complexity Measures and Methods*, Walter de Gruyter, Berlin.

Reviewer's biography:

David E. Peercy is a project leader on software-related projects at Sandia National Laboratories at Albuquerque, New Mexico in the USA. He currently chairs a software supportability project for the Society of Automotive Engineers (SAE). His prior assignments have included developing for the US Air Force Operational Test and Evaluation Center (AFOTEC) a software tool, methodology, and procedures for assessing risks associated with software maintenance. Dr. Peercy holds a Ph.D. in Mathematics from New Mexico State University. His e-mail address is: depeerc@sandia.gov.

[1] The Risk Assessment Methodology for Software Supportability (RAMSS) research work was performed for the Air Force Operational Test and Evaluation Center, Kirtland AFB, NM over a period from 1977 to 1989. The three above marked documents have been entered into and are available from the Defense Technical Information Center repository in Arlington VA.